REMARKS

Applicants have now had an opportunity to carefully consider the Examiner's comments set forth in the Office Action of June 28, 2004.

All of the Examiner's objections and rejections are traversed.

Reexamination and reconsideration are respectfully requested.

The Office Action

The drawings filed on November 5, 2001 stand accepted by the Examiner.

Claims 1-4, and 6 stand rejected under 35 U.S.C. §102(e) as being anticipated by Baker et al. (U.S. Application No. 2004/0163544 A1). In the Office Action mailed June 28, 2004, however, the Examiner made reference to "Baker et al. U.S. Patent 5,408,603." It is assumed by Applicants herein that this was an inadvertent error, in that U.S. Patent No. 5,408,603 was issued to Van de Lavoir, et al., and also, the remarks made by the Examiner were consistent with U.S. Application No. 2004/0163544 by Baker et al. (hereinafter Baker).

Claim 5 stands rejected under 35 U.S.C. §103(a) as being unpatentable over Baker as applied to claim 1, and further in view of U.S. Patent No. 5,212,635 issued to Ferriter.

Claims 7-9 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Baker and U.S. Patent No. 6,321,243 issued to Ballard.

The Art Rejections

In paragraph 1 of the Detailed Action, in rejecting independent claims 1 and 6, the Examiner cites the "move," "file," "eye," "globe," "money," "keypad" and "file cabinet" symbols of Baker as disclosed in paragraphs 39 and 40. The Examiner interprets the "move" and "file" symbols as denoting action, the "eye," "globe" and "money" symbols as denoting materials, and the "keypad" and "file cabinet" symbols as denoting instruments in the same sense that the present application claims glyph instructions, namely, action glyphs representing defined actions which are able to be undertaken by the person following the instructions, material glyph images representing materials which are includable in created instructions, and instrumentation glyphs representing instruments which are includable in the created instructions.

Applicants respectfully traverse the Examiner's interpretation of the above-

mentioned symbols in Baker. For instance, the "eye," "globe" and "money" symbols of Baker clearly do not represent materials, but are generally related to actions the user of a computer device wishes the computer to undertake. For example, in paragraph 70, it is stated that "the present application applies both to utilizing symbol sequences to enable application programs and to execute commands" and then goes on to describe how commands can be executed in response to a selection of an associated polysemous symbol sequence. Baker describes The commands as including movement type commands such as move up, move left, move down, move right, page up, page down, etc., as well as standard computer commands such as new, open, save, quit, undo, etc.

Paragraph 71 adds that "an application program itself can be enabled utilizing a symbol sequence." In particular, the "eye" symbol is provided as an example in paragraph 77 with reference to being included twice in an "about" command. Paragraph 39 discloses the "globe" or "all" symbol as being used in sequences which access the internet category of application programs, and also being used as a sub-category symbol in sequences accessing the global moves such as a "move to top" command. The Examiner has not provided any example in Baker that teaches or suggests the use of these or other symbols as claimed in the present claims. To the contrary, these symbols are used by Baker for invoking application programs or actions taken by application programs. In fact, as set forth in the abstract, Baker is directed towards a computer device, such as a mobile phone or a PDA that is able to access an application program or command based on the selection of corresponding sequences or symbols. Independent claims 1 and 6, on the other hand, are directed to a system "wherein selected ones of the action glyphs, material glyphs and instrumentation glyphs are arranged in relationship to each other in accordance with a predetermined structure to form a specific instruction understandable by the person following the instruction irrespective of the written language understood by the person," and a method of "arranging the selected glyphs in relationship to each other in accordance with a predetermined structure to form a specific instruction understandable by the person following the instruction irrespective of the written language understood by the person."

The Examiner further likened the "keypad" and "file cabinet" symbols of Baker as instrumentation symbols. Applicants disagree with the Examiner's interpretation of these symbols as well. Examples are provided in paragraph 39 for the "keypad"

and "file cabinet" symbols where the symbols are used for information programs such as "dictionary" ("find" and "find" and "keypad"), HanDBase ("find" and "find" and "file cabinet") and a calculator program ("control" and "control" and "keypad"). These symbols are used by Baker for invoking application programs or actions taken by application programs as further described in paragraph 57 wherein "selection of any one of these valid 'third' symbols in a symbol sequence will enable a particular application program." Claims 1 and 6 of the present application further clarify that the material glyphs are defined to represent materials which are includable as part of the instruction and the instrumentation glyphs are defined to represent instruments which may be included in instructions to the user.

In view of the foregoing arguments, the Examiner has not shown that Baker teaches the use of glyphs to represent either materials or instrumentation as recited in claims 1 and 6 of the present application. Applicants submit, therefore, that independent claims 1 and 6, and claims 2-5, depending from claim 1, are patentably distinct over the cited reference and in condition for allowance.

With reference now to independent claim 7, the Examiner again cites the "move" and "file" symbols as denoting action, the "eye," "globe" and "money" symbols as denoting materials, and the "keypad" and "file cabinet" symbols as denoting instruments in the same sense that the present application claims glyph instructions. Applicants traverse the Examiner's opinion in this regard for the same reasons provided above with reference to independent claims 1 and 6.

Additionally, however, in rejecting claim 7, the Examiner also cites the converting of inputted text to glyphs, as disclosed in Ballard (col. 1, lines 26-33, col. 3, lines 35-41, and col.7, lines 19-23), and likens this to the recited limitation of "a translator configured to receive the inputted instructions and to interpret the inputted instructions so as to select ones of the action glyphs, material glyphs and instrumentation glyphs which represent the inputted instructions." The glyphs disclosed by Ballard, however, do not comprise instructions but, rather, are only representations of inputted characters as set forth in col. 1, lines 27-28. Ballard discloses that the glyph may also represent a string of characters such as the lowercase letters "fi" (col. 1, lines 29-31), however, the example Ballard offers, is clearly intended to handle the well-known ligatures as known in the typesetting art, particularly since Ballard specifically cites the lowercase combination of the letters.

For the Examiner's convenience, a definition of "ligature" from the Merriam-Webster Online Dictionary has been provided. See also *Windows Glyph Processing: Part Two: Glyph Processing in Detail* at http://www.microsoft.com/typography/Glyph%20Processing/detail.mspx, also provided for the Examiner's convenience. In particular, the "fi" ligature is discussed on page 5 of 17, 2nd paragraph. It is well known in the art that the letters making up one of the known ligatures touch each other when correctly typeset, as opposed to having a small space between the letters. The glyphs in Ballard, are simply graphical representations of the corresponding letters, and Ballard is apparently providing the capability to represent ligatures as a single graphical entity. Ballard does not teach or suggest that the glyphs described therein are representations of instructions. The glyphs disclosed in the present application, on the other hand, as recited in claim 7, clearly represent instructions related to actions, materials, and/or instruments.

In view of the foregoing arguments, Applicants submit that independent claim 7, and claims 8-9, depending therefrom, are patentably distinct over the cited reference and in condition for allowance.

Prior art considered pertinent to the applicant's disclosure and made of record, but not relied upon by the Examiner, has been reviewed by the applicants. The applicants submit that these references alone or in combination do not teach the present invention.

CONCLUSION

For the reasons detailed above, it is submitted all claims remaining in the application (Claims 1-9) are now in condition for allowance. The foregoing comments do not require unnecessary additional search or examination.

No additional fee is believed to be required for this Amendment A. However, the undersigned attorney of record hereby authorizes the charging of any necessary fees, other than the issue fee, to Xerox Deposit Account No. 24-0037.

In the event the Examiner considers personal contact advantageous to the disposition of this case, he/she is hereby authorized to call Mark Svat, at Telephone Number (216) 861-5582.

Respectfully submitted,

FAY, SHARPE, FAGAN,
MINNICH & McKEE, LLP

11/29/04
Date

Mark S. Svat
Reg. No. 34,261
1100 Superior Avenue, 7th Floor
Cleveland, Ohio 44114-2579
(216) 861-5582

Attachments

N:\XERZ\200422\US\GTY0000263V001.doc

HOME
PREMIUM SERVICES
M-WCollegiate.com
M-WUnabridged.com
Britannica.com
Multi-User Licenses
DOWNLOADS
WORD OF THE DAY
WORD GAMES
WORD FOR THE WISE
ONLINE STORE
HELP
Merriam-Webster Inc. Company information

Merriam-Webster Online Dictionary

Thesaurus

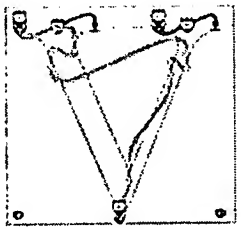
One entry found for **ligature**.

Main Entry: **lig a ture**
Pronunciation: `'li-g&-"chur, -ch&r, -"tur, -"tyur`
Function: *noun*
Etymology: Middle English, from Middle French, from Late Latin *ligatura,* from Latin *ligatus,* past participle of *ligare* to bind, tie; akin to Middle Low German *llk* band, boltrope, Albanian *lidh* I tie
1 a : something that is used to bind; *specifically* **:** a filament (as a thread) used in surgery **b :** something that unites or connects **:** **BOND**
2 : the action of binding or tying
3 : a compound note in mensural notation indicating a group of musical notes to be sung to one syllable
4 : a printed or written character (as *æ* or [ff]) consisting of two or more letters or characters joined together

For **More Information on "ligature" go to Britannica.com**

Get the **Top 10 Search Results for "ligature"**

Pronunciation Symbols



Developing fonts

Windows Glyph Processing: Part Two:Glyph Processing in Detail

On This Page

OpenType Font Tables in Detail

As the central element of Windows glyph processing, the OpenType font format is a vast subject. Fortunately, the specification provides not only minute detail of the font table structures, header formats and internal tag lists, but a large number of examples that demonstrate the function of specific aspects of the technology. The following information provides a detailed introduction to the key glyph processing aspects of the OpenType format, somewhere between the level of the overview in Part One and the technical coverage of the specification itself, and some 'real world' examples of OTL feature implementation.

OpenType is an extension of the original TrueType format, developed in partnership by Microsoft and Adobe Systems. OpenType makes use of the basic architecture of TrueType to add support for a wide variety of glyph substitution, positioning, justification and alignment features. The TrueType font format is based on a series of tables containing code and data for different aspects of the font architecture: character mapping, glyph outline descriptions, spacing metrics information, glyph naming, copyright and licensing, among others.

OpenType adds to the number of possible tables in the existing TrueType specification, to allow a greater degree of intelligence to be built into a font. Most of these new tables are optional, and it is important to note that it is possible to produce an OpenType font without any of the glyph substitution and positioning features encountered in Part One. The tables that will be discussed in this article are the five optional Advanced Typographic Tables:

- GDEF - Glyph definition data
- GSUB - Glyph substitution data
- GPOS - Glyph positioning data
- BASE - Baseline data
- JSTF - Justification data

and two required tables:

- CMAP - Character to glyph mapping
- DSIG - Digital Signature

There is also a discussion of the role of script and language system tags.

CMAP Table

Every glyph in a TrueType font is identified by a unique Glyph ID (GID), a simple sequential numbering of all the glyphs in the font. These GID's are mapped to character codepoints in the font's CMAP table. In OpenType fonts, the principal mapping is to Unicode codepoints; that is, the GIDs of nominal glyph representations of specific characters are mapped to appropriate Unicode values.

The key to OpenType glyph processing is that *not every glyph in a font is directly mapped to a codepoint*. Variant glyph forms, ligatures, dynamically composed diacritics and other rendering forms do not require entries in the CMAP table. Rather, their GID's are mapped in layout features to the GIDs of nominal character forms, *i.e.* to those glyphs that do have CMAP entries. This is the heart of glyph processing: the mapping of GIDs to each other, rather than directly to character codepoints.

In order for fonts to be able to correctly render text, font developers must ensure that the correct nominal glyph form GID's are mapped to the correct Unicode codepoints. Application developers, of course, must ensure that their applications correctly manage input and storage of Unicode text codepoints, or map correctly to these codepoints from other codepages and character sets.

GDEF Table

As discussed in Part One, the most important tables for glyph processing are GSUB and GPOS, but both these tables make use of data in the Glyph Definition (GDEF) table. The GDEF table contains three kinds of information in subtables: glyph class definitions that classify different types of glyphs in a font; attachment point lists that identify glyph positioning attachments for each glyph; and ligature caret lists that provide information for caret positioning and text selection involving ligatures.

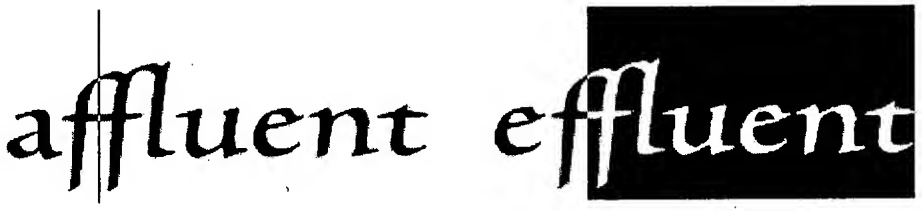
The Glyph Class Definition subtable identifies four glyph classes: simple glyphs, ligature glyphs (glyphs representing two or more glyph components), combining mark glyphs (glyphs that combine with other classes), and glyph components (glyphs that represent individual parts of ligature glyphs). These classes are used by both GSUB and GPOS to differentiate glyphs in a string. For example, to distinguish between a base vowel (simple glyph) and the accent (combining mark glyph) that a GPOS feature will position above it.

The Attachment Point List identifies all the glyph attachment points defined in the GPOS table. Clients that access this information in the GDEF table can cache attachment coordinates with the rasterized glyph bitmaps, and avoid having to recalculate the attachment points each time they display a glyph. Without this table, GPOS features could still be enabled, but processing speed would be slower because the client would need to decode the GPOS lookups that define the attachment points and compile its own list.

The Ligature Caret List defines the positions for the caret to occupy in ligatures. This

caret to step across the component characters of a ligature, and for the user to select text, including parts of ligatures.

In the example on the left, below, the caret is positioned between two components of a ligature; on the right, text is selected from within a ligature.



The Ligature Caret List can contain positioning data in both X and Y directions.

GSUB Table

The GSUB table contains substitution lookups that map GID's to GID's and associate these mappings with particular OpenType Layout features. The OpenType specification currently supports six different GSUB lookup types:

1. Single
   Replaces one glyph with one glyph.
2. Multiple
   Replaces one glyph with more than one glyph.
3. Alternate
   Replaces one glyph with one of many glyphs.
4. Ligature
   Replaces multiple glyphs with one glyph.
5. Context
   Replaces one or more glyphs in context.
6. Chaining Context
   Replaces one or more glyphs in chained context.

Although these lookups are defined by the font developer, it is important for application developers to understand that some features require relatively complex UI support. In particular, OTL features using Type 3 lookups may require the application to present options to the user (an example of this is provided in the discussion of OTLS in Part One). In addition, some registered features allow more than one lookup type to be employed, so application developers cannot rely on supporting only some lookup types. Similarly, features may have both GSUB and GPOS solutions—*e.g.* the 'Case-Sensitive Forms' feature—so applications that want to support these features should avoid limiting their support to only one of these tables. In setting priorities for feature support, it is important to consider the possible interaction of features and to provide users with powerful sets of typographic tools that work together.

Scripts and Language Systems

importance of script and language systems. These are a central aspect of the OpenType format: all glyph processing is defined within a context of script and language system. Even features that are not script or language specific by nature are located in this hierarchy:

```
Script > Language System > Feature > Lookup
```

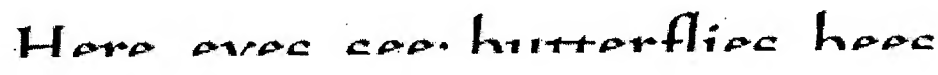
An OpenType language system tag is always associated with a script tag, and indicates a specific orthographic convention for writing that language in that script. Because many written languages sharing a common script also share common typographic requirements, font developers can specify a Default <dflt> language system for each script that will be sufficient to support all but a few languages. By specifying additional language systems, the font developer can modify the function of OTL features when a script is used to represent these languages. This, of course, requires application level support for different language systems, possibly including mapping of OTL language systems to user locales, as well as language tagging of text runs.

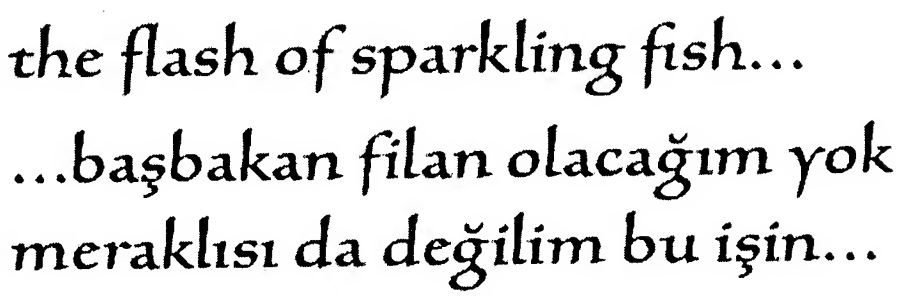
In this example of a truncated tree structure, a Turkish language system has been added to Latin script support in a font in order to enable an exception to the default results of the 'Standard Ligatures' feature. This feature includes the common f-ligatures in the Default language system, including the *ffi* and *fi* ligatures, but the latter are excluded in the Turkish language system to avoid confusion between the dotless *i* in the ligature glyphs and the Turkish letter with the same form. Mnemonic glyph names are used here, rather than GID numbers.

```
Latin <latn>
Default <dflt>
        Standard Ligatures <liga> (GSUB feature)
        All f-ligs (GSUB type 4 lookups)
                f f i -> ffi
                f f l -> ffl
                f f -> ff
                f i -> fi
                f l -> fl
Turkish <TRK>
        Standard Ligatures <liga> (GSUB feature)
        ff and fl f-ligs (GSUB type 4 lookups)
                f f l -> ffl
                f f -> ff
                f l -> fl
```

It should be noted that there are several different ways to achieve the same rendering, using different sets of lookups within the OTL feature. The best method will depend on the nature of the different languages supported by the font, and the number of exceptions to the default glyph processing.

The differing results of the same feature applied under two different language systems are shown below. [10]





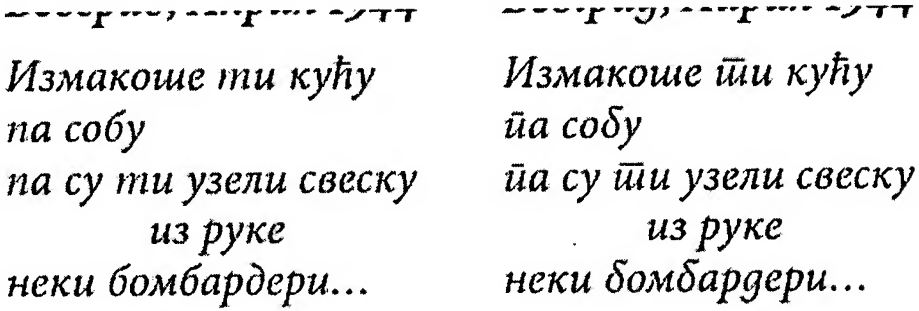
> **NOTE:** Although they enable exceptions to the Default language system feature behavior, additional language systems do not act exceptionally; that is, *all* desired features need to be associated with each language system, not just those that differ from the Default language system feature set.

While any OTL feature can be associated with different language systems, and may provide distinct glyph processing results for each, some registered features are designed specifically to take advantage of OpenType's language system tags. One of these is the 'Localized Forms' feature that associates stylistic glyph variants with particular language systems. This enables developers to provide support for different localized typographic cultures within the same script, and to 'disunify' Unicode codepoint assignments at the font level. One important implementation of this would be in East Asian fonts that provide preferred forms of Han ideographs for Traditional and Simplified Chinese, Japanese and Korean users.

In the simpler example below, the 'Localized Forms' feature is used in an italic font to substitute the traditional Serbian forms of some Cyrillic letters where they differ from the international norms based on the Russian tradition. Again, mnemonic glyph names are used rather than GID's.

```
Cyrillic <cyrl>
Default <dflt>
        ...
Serbian <SRB>
        Localized Forms <locl> (GSUB feature)
        Serbian forms (GSUB type 1 lookups)
                cyrbe -> cyrbe.serb
                cyrghe -> cyrghe.serb
                cyrde -> cyrde.serb
                cyrpe -> cyrpe.serb
                cyrte -> cyrte.serb
```

The illustration below shows the dramatic results of this feature applied to just a few lines of Serbian poetry. [11] The text on the left uses the font's default glyph forms, the Russian norms, and the text on the right uses the substituted Serbian forms; the affected letters are indicated in red. Once again, the text string remains entirely unchanged, and only the rendering differs.



As in the previous example, there are many ways to achieve the same results, and the OpenType specification does not favor any particular solution. It would be perfectly legitimate to designate the Serbian forms as the default glyphs for these Cyrillic characters, *i.e.* to make Serbian the Default language system. However, this would require many more instances of the 'Localized Forms' feature to be applied to substitute the Russian forms, that are also used for so many other languages like Byelorussian, Ukrainian and dozens of minority languages within the Russian Federation. OpenType font developers need to plan their projects carefully to make efficient use of features and lookups.

GPOS Table

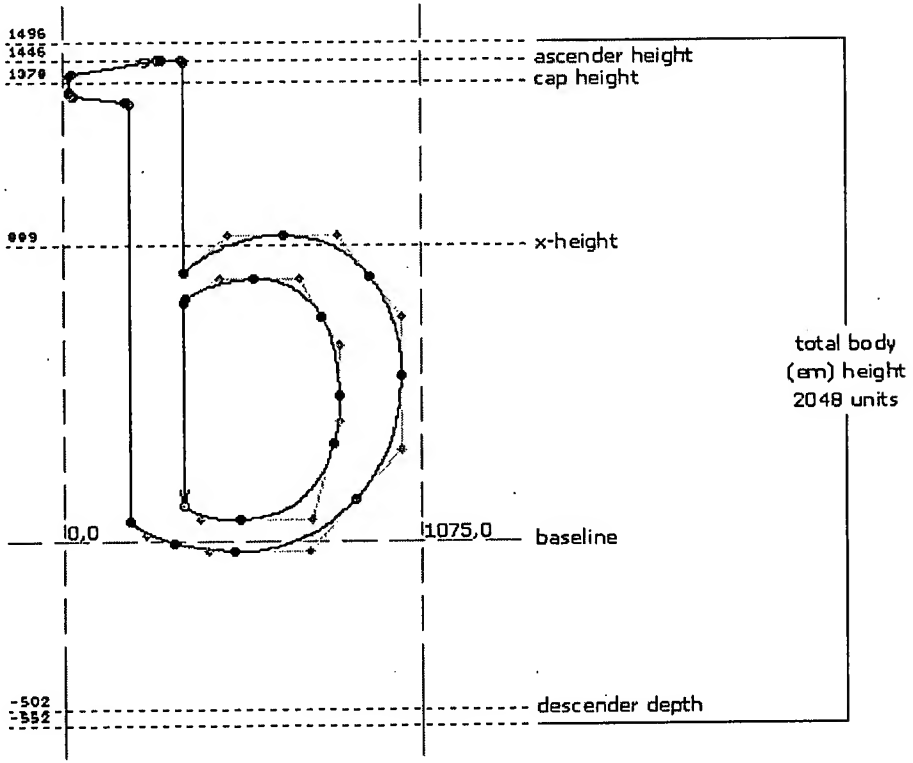
The GPOS table contains a very powerful set of lookup types to reposition glyphs relative to their normative positions and to each other. Glyph positioning lookups work in two ways: by adjusting glyph positions relative to their metrical space or by linking predefined attachment points on different glyphs. These two methods are further divided into specific adjustment and attachment lookup types that can be used to control positioning of diacritics relative to single or ligatured characters, and even to enable chains of contextual positioning operations. The OpenType specification currently supports eight different GPOS lookup types:

1. Single adjustment
   Adjusts the position of a single glyph.
2. Pair adjustment
   Adjusts the position of a pair of glyphs.
3. Cursive attachment
   Attaches cursive glyphs to each other.
4. MarkToBase attachment
   Attaches a combining mark to a base glyph.
5. MarkToLigature attachment
   Attaches a combining mark to a ligature.
6. MarkToMark attachment
   Attaches a combining mark to another mark.
7. Context positioning
   Positions one or more glyphs in context.
8. Chained Context positioning
   Position one or more glyphs in chained context.

Adjustment lookups are defined using the font's internal metrical units (font units), which are specified as units of the total body or em height of the font, hence em units. The TrueType specification allows font developers to set the number of units per em in a font, but recommends the use of a power of two. This is actually a procurement requirement for fonts that ship with Microsoft products, and the majority of TrueType fonts have an em of 2048 units.

It was much easier to visualize an em in earlier days, when the body of a piece of type was a chunk of metal. Digital glyph outlines are drawn on a Cartesian grid originating at the intersection of the left sidebearing and the nominal baseline. Digital ems are invisible most of the time, and the same glyph in different fonts may occupy more or less of the body height, and may even overflow the total height.

The illustration below shows the lowercase *b* from the Windows 2000 font **Sylfaen** on its Cartesian grid, with an em of 2048 units. **Sylfaen** is a font that comes close to filling the em height with its Latin ascender height and descender depth, but there are many fonts that are 'cast small on the body', whose glyphs occupy much less vertical space.



GPOS adjustment lookups can shift the position of a glyph in both X and Y directions, and can move a glyph well beyond its normative sidebearings and em height. Lookups are defined as offsets from the normative glyph position

Among the OTL features that call GPOS table entries, the 'Kerning' feature deserves special mention because it provides a new solution for an existing function in most text layout software. This feature generally implements type 2 GPOS lookups (chained contextual kerning is also possible using type 8 lookups), and has several advantages over traditional kerning pairs as stored in the TrueType KERN table. The GPOS 'Kerning' feature can make use of class-based kerning, that is, adjustment of inter-glyph spacing in horizontal text according to predefined classes of glyphs with similar shapes and spacing. This both speeds up font production, especially in OpenType fonts with large glyph sets involving many diacritic variants of base glyphs, and decreases the file size of a well-kerned font.

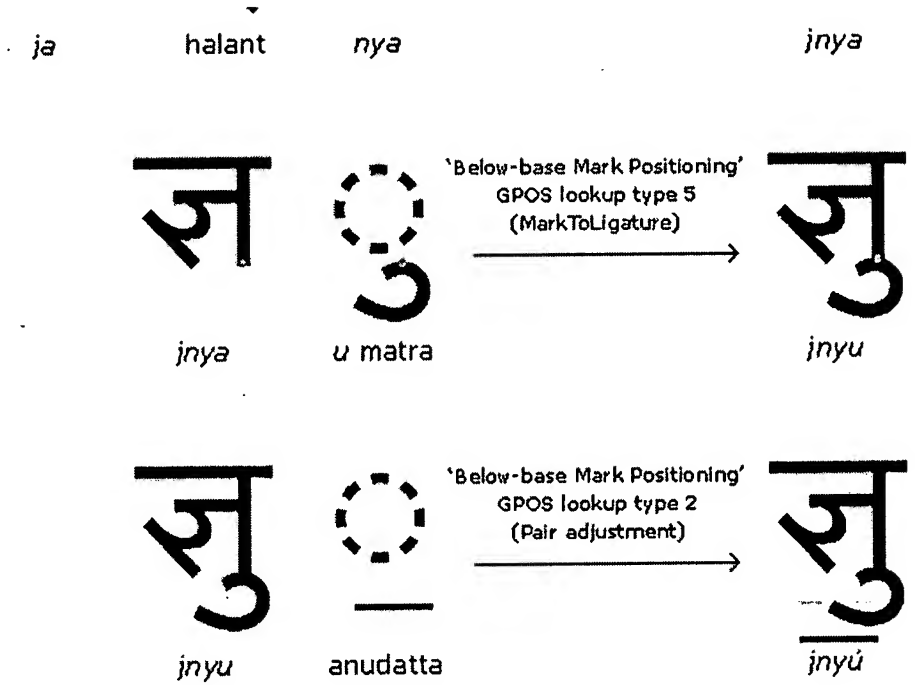
The 'Kerning' feature can also provide device-dependent kerning data for specific bitmap sizes to optimize screen typography. GPOS kerning and KERN table kerning information can be stored in the same font, and applications can decide which to make use of, although the production advantages of class-based kerning may render the KERN table effectively obsolete as more applications support the 'Kerning' feature.

GPOS attachment lookups are made by predefining one or more absolute points on a glyph's em grid, and then aligning attachment points on different glyphs. Attachment points are defined in the Glyph Definition Table (GDEF), which is referenced by the GPOS table. GPOS lookup types allow for attachment of marks, *e.g.* combining accents or vowel matras in Indic scripts, to base glyphs or to other marks. The latter MarkToMark attachments can be used to avoid collisions when more than one mark is applied to a base glyph. When contextual positioning or chained contextual positioning is used, it is possible to reposition glyphs more than once as the sequence of input characters is rendered.

The following example, using the Windows Devanagari UI font, **Mangal**, demonstrates how two different GPOS lookups are applied in conjunction with an initial GSUB lookup in shaping an Indic syllable. In the first line, the *ja* consonant, its inherent vowel suppressed by the halant, combines with the *nya* consonant to form an akhand, a required ligature form. This is achieved by implementing a type 4 (Ligature) GSUB lookup called by the 'Akhand' feature.

In the second line, the inherent vowel in *jnya* is replaced by the *u* vowel matra. This is a type 5 (MarkToLigature) GPOS attachment lookup called by the 'Below-base Mark Positioning' feature. The attachment point on the two glyphs is indicated in the illustration by a small green dot. The lookup aligns these attachment points to correctly position the *u* matra below the conjunct ligature.

In the third line, an anudatta stress accent is added to the syllable; this is lowered from its normative position and repositioned below the *u* matra by a type 2 (Pair adjustment) GPOS lookup, also called by the 'Below-base Mark Positioning' feature. The pre-adjustment position of the anudatta is indicated in the final syllable by the pale gray rectangle.



BASE Table

BASE table entries, defined relative to the em height, are used to adjust the vertical position of lines of text composed with glyphs of different scripts and point sizes. Every glyph in a font has a nominal default baseline and, presuming the font has been well made, this baseline will provide consistent alignment appropriate for the supported script or scripts. When scripts are mixed in text, correct alignment and interlinear spacing may require an adjustment of the vertical position of one or more scripts relative to the baseline of the 'dominant' script.

In the example below, the dominant script is Latin, and the single kanji character needs to be moved down in order to align it correctly with the Latin text. CJKV ideographs [12] have a default baseline near the bottom of the character, and in order to properly align with Latin text they need to have a BASE table entry identifying a second baseline that will set them lower on the line of text. In the first sample, the kanji character's default baseline is used; this positions the glyph too high and it almost collides with the descender of the *p*. In the second sample, this problem has been corrected by using the BASE table entry for dominant Latin text.

The second repetition of the

verb bestow, 贈 *zō sō*, adds to
the irony of the poet's loss...

The second repetition of the
verb bestow, 贈 *zō sō*, adds to
the irony of the poet's loss...

In addition to allowing multiple baselines to be identified for different scripts, the BASE table also gives clients the option of using minimum and maximum text extent entries to override the default text extent of particular scripts. The BASE table can define script, language system and feature specific min/max extent values. This is particularly useful for word processing applications that employ automatic interlinear adjustment to prevent collisions, as layout can be optimized for specific glyph processing features.

JSTF table

The JSTF table provides font developers with increased control over glyph substitution and positioning in justified text, *i.e.* text that is flush to both the left and right margins. JSTF table entries are designed to supplement an application's justification algorithms by enabling or disabling specific OTL features. The table entries present a sequence of suggested priorities to improve the spacing and general appearance or 'color' of justified text, the options for which will depend on particular script and language systems. For example, the spacing of justified Latin text might be improved in some circumstances by decomposing ligatures, while Arabic text may benefit from the use of swash forms or kashidas. [13]

DSIG table

At the beginning of this section, I referred to the DSIG table as a 'required' table. In fact, a digital signature is *not* required, in the same sense that the CMAP and many other tables are required, simply for the font to work. A font without a DSIG table will work, but it will not be recognized as an OpenType font by the Windows operating system. Because the OpenType format is an extension of the TrueType format, and most of the new tables are optional, Windows makes a distinction between the two formats based solely on the presence or absence of a DSIG table. A font with a DSIG table will be recorded in the Windows font folder as an OpenType font and presented with the OpenType icon.

A digital signature assures users that the signed software or document (in this case, the font) has not be altered or tampered with since it was signed by the maker, and that it does not contain malicious code or dangerous flaws. Digital signatures are based on a

Font developers can download a free font signing tool here. This is a command line utility that, in addition to adding a DSIG table to a font, runs a number of glyph integrity checks to confirm that the font is minimally conformant with the font format specification. If the font contains errors that may affect its performance, the digital signature tool will fail.

OpenType Resources

The OpenType specification is published online, along with the basic TrueType specification. This page also includes the document Creating and supporting OpenType fonts for Indic scripts.

Microsoft recently released its Visual OpenType Layout Table tool, VOLT, for adding GDEF, GSUB and GPOS tables to fonts. VOLT is available under a free license. More information can be found at the Typography VOLT page, including release notes, tutorials and links to the very active international VOLT user community on MSN.

VOLT takes advantage of Microsoft's Typography Integrated Development Environment, TIDE, an integrated application suite that enables font tools to leverage a shared infrastructure. TIDE is available under a free license to font tool developers. More information can be found at the Typography TIDE page.

Detailed information about digital signatures for fonts and the free signing tool is available at the Typography Digital Signatures page.

↑ Top of page

OTLS in Detail

The OpenType Layout Services library is currently available under a free license to application developers who are interested in using its helper functions to implement OpenType glyph processing. As explained in Part One, OTLS is composed of helper functions that insulate client applications from the details of the OpenType font tables and assist in applying GSUB and GPOS features. Client applications are free to make full or partial use of OTLS functions, making it a highly adaptive solution for developers who want to implement OpenType glyph processing support within existing text layout architecture.

In order to process runs of text, as defined in Part One, a client needs to be able to store characters, glyph coordinates, and flag and formatting properties. To make this easy, OTLS provides a general purpose object called an otlList. In addition to identifying and tagging runs of text, an OTLS client needs to be able to create and fill otlLists using inline helper functions. These otlLists will be used as input and output parameters for OTLS functions.

The principal OTLS functions are grouped in three categories. *Font Information Functions* are used to query an OpenType font about what scripts, language systems and features it supports:

- GetOtlVersion

- GetOtlScriptList
  Enumerates scripts in a font.
- GetOtlLangSysList
  Enumerates language systems in each script.
- GetOtlFeatureDefs
  Enumerates OTL features in each language system.

Text Information Functions return information about text layout and locate run elements such as character position and feature parameters:

- GetOtlLineSpacing
  Returns interlinear spacing for a run of text.
- GetOtlBaselineOffsets
  Returns baseline adjustment between two scripts in adjacent runs (using font BASE table).
- GetOtlCharAtPosition
  Identifies what character is at given x,y coordinate.
- GetOtlExtentOfChars
  Returns location of character range.
- GetOtlFeatureParams
  Finds feature parameters within a run.

Text Layout Functions implement GSUB and GPOS features:

- SubstituteOtlChars
  Performs primary glyph substitutions, i.e. *substitutions of default glyphs.*
- SubstituteOtlGlyphs
  Performs secondary glyph substitutions, i.e. *substitutions of subsequent glyphs.*
- PositionOtlGlyphs
  Performs initial glyph positioning.
- RePositionOtlGlyphs
  Adjusts glyph positioning.

In addition to these functions, OTLS provides a resource management function, FreeOtlResources, to clear client memory.

It should also be noted that, although it pays particular attention to Windows glyph processing, the OTLS library is designed to be adaptable to other platforms. Developers who are interested in obtaining a version of OTLS for a non-Windows platform should contact Microsoft Typography.

Licensing information for OTLS is available here. The OTLS library and documentation is distributed through an MSN web community for licensed users.

↑ Top of page

Uniscribe in Detail

Explorer 5.0+, which is the current update mechanism. Uniscribe is not available under license to other vendors, but any application and font needing complex script shaping in Windows 2000 can make use of it. Applications can also use Uniscribe to display and print complex script text on older versions of the operating system, if the DLL is present.

Windows applications have a number of system API options for performing text layout. These include basic Win32 text APIs such as **TextOut**; more advanced Win32 edit controls that have been extended in Windows 2000 to support multilingual text and some aspects of complex scripts, such as right-to-left reading order; the higher level interfaces of RichEdit control that take advantage of Uniscribe; and finally Uniscribe itself, which can be called directly by clients.

Uniscribe provides a large set of API's to handle all aspects of text layout for supported scripts, including cursor position and hit testing, advance width calculation, linebreaking, complex script determination, localized digit substitution, etc.. The full list of API's in Uniscribe is available at the MSDN online library. A review of these will provide developers with a sense of scope of the processor's functions.

We have already seen, in our Devanagari example in Part One, three of these API's in action. Uniscribe divided the text run into clusters and generated glyphs (ScriptShape function); these glyphs are then positioned (ScriptPlace function) and displayed (ScriptTextOut function). During this process, additional API's, such as **ScriptStringValidate**, might be called to confirm processing requirements.

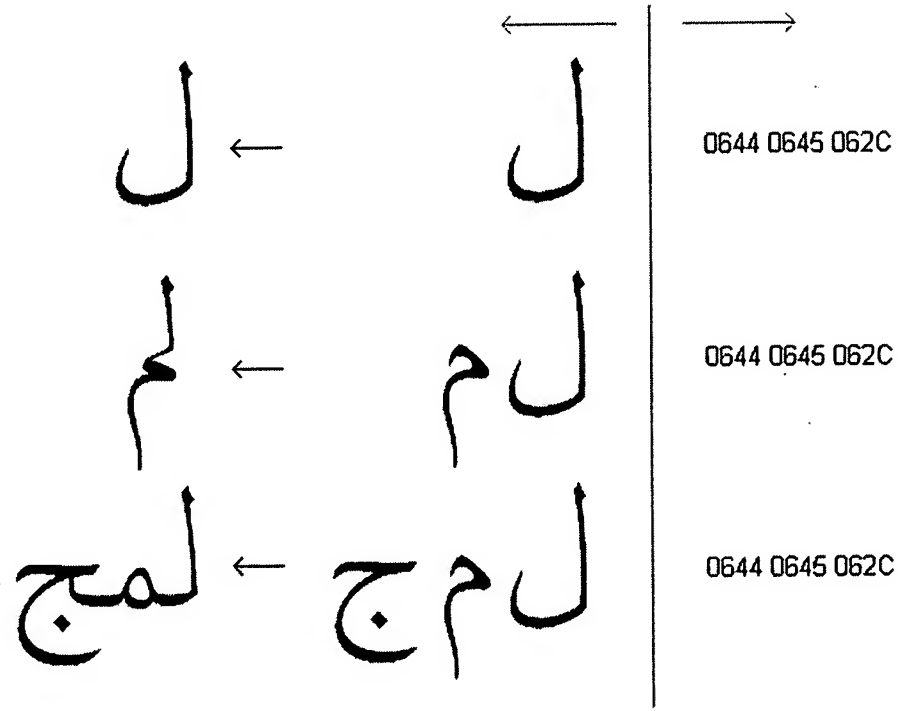
The **ScriptShape**, **ScriptPlace** and **ScriptTextOut** API's all interact with the Uniscribe shaping engines. As discussed in Part One, each shaping engine contains specific knowledge about a script or groups of related scripts. As support for new scripts is added to Uniscribe, new shaping engines may be defined or existing shaping engines may be extended to cover related scripts. For example, Syriac script support has been added to the existing Arabic shaping engine, while the similar processing requirements of Hebrew and Thaana allow the latter to be added to the Hebrew shaping engine.

All the script engines analyze text runs to isolate the basic unbreakable element, the cluster. As we saw in the Devanagari example, clusters identified by the Indic shaping engine correspond to syllables. In Arabic, each cluster corresponds to an adjacent pair of characters, and the shaping engine moves along the text string classifying each character relative to its neighbors. For example, the first Arabic letter (determined by Unicode character properties) following a space character will first be classified as an isolated form, but if the next character is also an Arabic letter, the first will be reclassified as an initial form and the second classified as a final form. The shaping engine then moves along the string, making this second letter the first character of the next cluster. If the following character is a third letter, the second is reclassified as a medial form, the new character is classified as a final form, and the engine moves on.

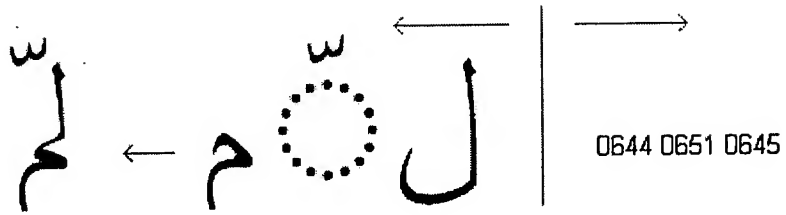
Uniscribe calls OTLS functions to render the correct forms using the OpenType Layout GSUB features 'Initial Forms', 'Medial Forms' and 'Terminal Forms' (the isolated form is presumed to be the default glyph form). This will result in basic minimum Arabic shaping.

In the illustration below, a ligature feature has been applied to render the cluster in the

cluster, so it goes back and replaces the ligature form from line two with the initial and medial forms of the first and second letters, to which the new final form letter is joined. The Unicode codepoints to the right of the grey line are stored in logical order, left-to-right, while the Arabic glyphs are rendered right-to-left. The cluster being processed in each line is indicated by the red codepoints.



The Arabic shaping engine will automatically call all required OTL features, *e.g.* the 'Required Ligatures' feature to render the *lam alef* combination. A client application can make optional features available to users; for Arabic typography, these will likely include additional ligatures and swash forms. If vowel points or other marks are included in a text string, the Arabic script engine will use the Unicode character properties and shaping rules to ensure that these do not interfere with the rendering of the correct letter forms. GPOS lookups can be used to dynamically and contextually reposition point marks.



Note that, in addition to the complex script support described here, Uniscribe also understands the non-OpenType layout font formats for Arabic, Hebrew and Thai that were supported in previous localized versions of the operating system.

It is worth repeating that Uniscribe script engines implement Unicode Standard shaping rules for complex scripts. This gives font and application developers a common set of complex script expectations: font developers should be able to expect an application to be able to execute—directly or by calling Uniscribe—script rules as defined in the Unicode Standard, and application developers should be able to expect fonts with glyphs and layout features that are responsive to these rules.

Uniscribe currently has shaping engines for the following scripts:

- Arabic
  Arabic, Syriac
- Hebrew
  Hebrew, Thaana
- Indic
  Bengali, Devanagari, Gujarati, Gurmukhi, Kannada, Malayalam, Oriya, Tamil, Telugu
- Old Hangul
  Hangul Jamo
- Thai
  Thai

Note that not all of these scripts are actively supported in Windows 2000 yet, usually due to the absence of fallback fonts. Fonts are in development for most of these scripts, and are in the pipeline to be added to Windows along with appropriate keyboard drivers, locale definitions, etc..

In addition to script support, Uniscribe provides processing for Unicode surrogate pairs that extend the number of characters that can be encoded in Unicode to almost one million. Among the first characters to be added to the surrogate extension planes are a large number of additional Han ideographs, so surrogate support will be an important aspect of East Asian text processing.

Uniscribe also implements the Unicode bidirectional (bidi) algorithm for mixed and nested text directions. This is essential for correctly rendering Unicode text strings containing characters with different direction properties, *e.g.* Arabic words within English text. Use of the bidi algorithm is not limited to mixing scripts, however; Arabic and Hebrew are both scripts with strong right-to-left reading direction, but require digits to be rendered from left-to-right.

More information about Uniscribe, including technical documentation, is available from the MSDN library here.

The Microsoft Typography website provides an article, originally written for *Microsoft*

Systems Journal, entitled 'Supporting multilanguage text layout and complex scripts with Windows 2000'. This article provides additional information about Uniscribe, RichEdit and the Win32 edit controls as they relate to developing international software.

Notes

10. Both the English and Turkish samples are from the work of the Turkish poet Nazim Hikmet.

11. These are the opening lines of 'Belgrade, April 1944', by the Yugoslav poet Miodrag Pavlović. The opening of the poem is translated, by Bernard Johnson: *The bombers | removed your house | and your room | and took the notebook | out of your hand....*

12. CJKV is a common software development abbreviation for Chinese, Japanese, Korean and Vietnamese. It is also frequently encountered simply as CJK, since the Chinese ideographs are no longer part of the day-to-day writing system of most Vietnamese. The best source of information on Far Eastern text and digital typography is Ken Lunde's book, *CJKV Information Processing*.

13. A kashida is a lengthening stroke that extends, often very dramatically, certain Arabic letters in traditional calligraphic styles and fine typography.

Next section: conclusion

introduction | overview | detail | conclusion

Last updated 16 January 2001.

Top of page